# A Manual for a ModelChecker for Stochastic Automata

David Akehurst, Howard Bowman, Jeremy Bryans and John Derrick

March 1, 2001

## 1 Introduction

This technical report describes a modelchecker for Stochastic Automata, which was built based on the theory described in [BBD00].

The tool is available from:

http://www.cs.ukc.ac.uk/people/staff/dha/index.html

It accepts a stochastic automaton, a "timed probabilistic until" formula pattern and a time step parameter. Note that we do not yet allow adversaries, a clock which guards two or more transitions is considered a (run-time) error. Also, we have not yet implemented the full range of the temporal logic in [BBD00]; only the "timed probabilistic until" queries are allowed, and propositions must be atomic.

The algorithm will return one of three results: true, false or undecided. If it returns true, then the automaton models the formula. If it returns false, then the automaton does not model the formula. If it returns undecided, then the algorithm was unable to determine whether the automaton models the formula.

In the next Section, we briefly describe the notions of stochastic automata and temporal logic that we will use, then show how to use the tool in Section 6.

## 2 Stochastic Automata and Temporal Logic Expressions

A stochastic automaton is a structure $SA$ made up of

- a set of locations ($\mathcal{S}$) which includes an initial location $s_0$.
- a set of transitions ($\rightarrow\!\!\blacktriangleright$) between locations, where each transition has a clock and an action associated with it.
- a set of clocks ($\mathcal{C}$). Each $x \in \mathcal{C}$ is a random variable with *distribution function* $F_x$.

- a set of actions ($a$).
- a clock setting function ($\kappa$) which associates clocks to particular locations.

Clocks can only be associated with transitions going out from the locations they are set in. When the SA enters a location, all the clocks associated with that location are set according to their respective distribution functions. Clocks are set to positive values, and count down. When a clock reaches zero, the associated outgoing transition fires, and the moves to the next location. If there is no associated outgoing transition, nothing happens.

We will use a slightly simplified form of the stochastic automata, simplified in the following ways.

- each clock has a lower bound on the range to which it may be set;
- each clock has an upper bound on the value to which it may be set;
- all clocks set in a state must be consumed by at most one transition from that state[1];
- there is one clock on each transition.

We also associate atomic propositions with locations; these are used by the Temporal Logic Expressions.

### 2.1 Temporal Logic Expressions

In the technical report we use a simple timed stochastic temporal logic, including some simple propositional operators and a "timed probabilistic until" formula pattern. Here, we have implemented the algorithm only for the timed probabilistic until pattern below:

$$[a0\ \mathcal{U}_{<t}\ a1] > p$$

where $a0$ and $a1$ are propositions, $t$ is a time and $p$ is a probability value.

## 3 Entering Stochastic Automata

### 3.1 Probability Distribution Functions

Probability Distribution Functions are entered as JAVA class files, and stored under the directory `pdfs`. A simple example is the implementation of the function $F_X$

$$F_X(t) \quad = \quad \begin{cases} 0, & \text{if } t < 3 \\ 2(t-3) - (t-3)^2, & \text{if } t \in [3,4] \\ 1, & \text{if } t > 4 \end{cases}$$

which is implemented in JAVA as

[1] This differs from the tech report, where at least one trasnition must consume each clock, and means we don't have to worry about adversaries.

```
package pdfs;

public class PDFX
  implements IProbabilityDistributionFunction
{
  // implements F_X(t) =
  //    0                , if t < 3
  //    2(t-3) - (t-3)^2 , if t in [3,4]
  //    1                , o/w

  public Prob eval(double t) {
    Prob p = new Prob();
    if (t >= 0.0 && t < 3.0 ) {
        p.setValue(0.0);
    } else if (t >= 3.0 && t <= 4.0) {
        p.setValue( 2.0*(t - 3.0) - Math.pow(t - 3.0, 2.0));
    } else if (t > 4.0) {
        p.setValue(1.0);
    }
    return p;
  }
  public double max() { return 4.0; };
}
```

It is the users responsibility to ensure that the PDF restrictions are met; otherwise runtime errors may result.

Entering Stochastic Automata and Temporal Logic Expressions is through the GUI, called by the command `java ui.mcEditor`.

## 3.2 Installation

1. Unzip the zip file. This should create the following directories

   docs

   examples

   images

   jars

2. Compile the existing pdf files

   javac pdfs/*.java

3. Under unix to run the modelchecker user interface from the directory in which it has been installed. Ensure that the java VM version 1.2 is in the execution path, then type:

   modelchecker.bash

4. Alternativly, source the 'setClasspath.bash' script file passing the installation directory as a command line argument, e.g.

   source setClasspath.bash /dha/modelchecker

   Then run

   java ui.mcEditor

5. Under Windows, from a Command prompt, execute the setClasspath.bat file passing the installation directory as an argument, e.g.

   setClasspath.bat c:\pkgs\modelchecker

   then run

   java ui.mcEditor

## 3.3 Using the GUI

### 3.3.1 Entering locations

To enter a location, click on the location icon (second button), then click and drag to construct location. To enter the name and the propositions, double click near the top of the location, then type the name in the box, followed by the propositions in brackets separated by commas. For example, `s0, (On, Stopped)` indicates that in the initial location the propositions `On` and `Stopped` hold.

To enter the name of the associated clocks and PDFs, double click near the bottom of the location, then type the clock names and PDFs in the box. For example `f<PDFf>, g<PDFg>` indicates that the values of clock `f` are drawn from the stochastic function `PDFf`, and similarly for clock `g`.

## 3.4 Transitions

To enter a transition, click on the transition icon, (third button), then left click to start the transition, left click left to place anchor points and right click to finish. To enter the name of the transition and associated clock, type the name followed by set brackets containing the clock name, in the box that appears with the arrow. For example `arrow, {x}` indicates that the trasnition labelled `arrow` is guarded by the clock called `x`.

# 4 Temporal Logic Expressions

To enter a Temporal Logic Expression, click on the `TLE` icon. This brings up a new window with five boxes, labelled $a0, t, a1, p$ and $delta$. Enter the propositions in the boxes labelled $a0$ and $a1$. Box $t$ takes the time value, and box $p$ takes the probability value.

The box labelled *delta* takes the accuracy value discussed in [BBD00]. The smaller this value, the more accurate the final result can be. The restrictions on *delta* are: it must be smaller than the smallest possible value to which any clock can be set, and smaller values of delta cause the modelchecker to take longer.

# 5 Dispaying the SA and the TLE at the commandline

The forth icon (`Show SA`) displays a text discription of the SA, broken into locations. The discription of each location includes the location name, the list of propositions, the list of clocks and the list of outgoing transitions.

The sixth icon (`show TLE`) displays the until formula at the commandline, together with the value for delta.

# 6 Checking the model

The icon marked `check` checks the model against the formula. It returns the `pass/fail/undecided` result that the algorithm generates, and also gives values for the `totalpass`, `totalfail` and `error` probabilities that the algorithm uses. These can be used to make more informed judgements about the results. For example, if the result was `undecided`, but the `error` was large, then perhaps the same query using a smaller delta would produce a conclusive result.

# References

[BBD00] Howard Bowman, Jeremy Bryans, and John Derrick. A model checking algorithm for stochastic systems. Technical Report 4-00, University of Kent at Canterbury, 2000.